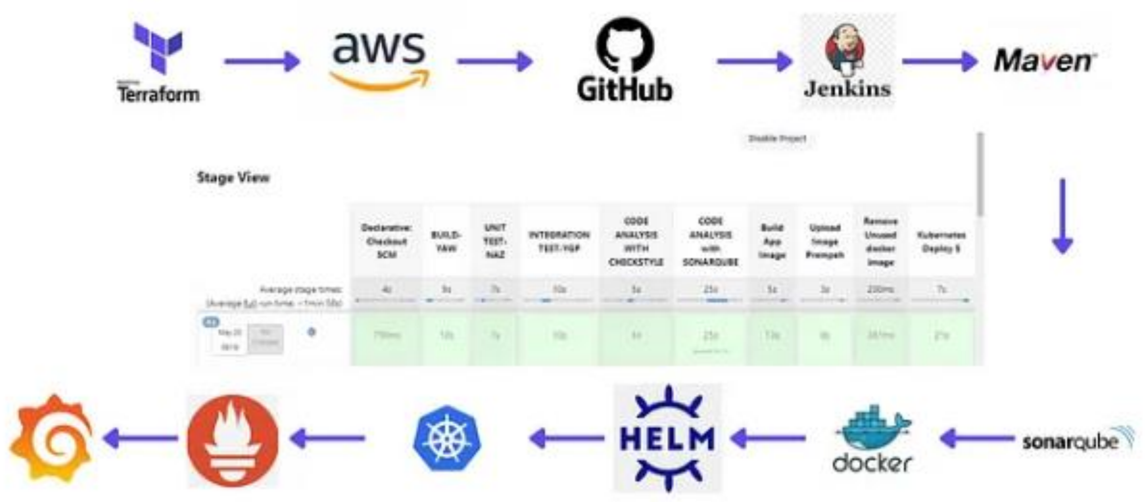
ULTIMATE DEVOPS PROJECT: E-COMMERCE APP

E-COMMERCE APPLICATION FOR ONLINE SHOPPING

Project Overview & Description



This project is a multi-tiered e-commerce application built with Spring Boot, hosted on AWS EC2 instances. The infrastructure is provisioned using Terraform, including Nginx (web layer), Tomcat (app layer), RabbitMQ (message broker), Memcache (cached layer), and MySQL (database tier).

The CI/CD pipeline, managed by Jenkins, automates building, testing, and deploying the application, with code stored in GitHub and Docker images in Docker Hub. Jenkins integrates several plugins, including SonarQube for code quality, Maven for build automation, and Docker for containerization.

Kubernetes, managed via Kops and Helm, orchestrates containerized deployments, ensuring scalability and reliability. Prometheus monitors system metrics, and Grafana visualizes these metrics on dashboards, facilitating real-time monitoring and troubleshooting.

This architecture ensures high availability, performance, and efficient resource utilization. Automated workflows and comprehensive monitoring help maintain system health, proactively identify, and resolve issues, and ensure a seamless user experience. The setup supports continuous delivery and integration, allowing for rapid and reliable updates to the application.

DETAILED ARCHITECTURE DESCRIPTION

✓ <u>Infrastructure Layer:</u>

This layer manages the foundational resources of the application using Terraform on AWS. Hosts various components on AWS EC2 instances. Terraform was used for Infrastructure as Code (IaC) to provision and manage the AWS infrastructure. This includes EC2 instances, VPCs, subnets, security groups, and other necessary AWS resources. Different AWS EC2 Instances host the different servers of the application including Jenkins server, SonarQube service, and the anchor server for Kops cluster.

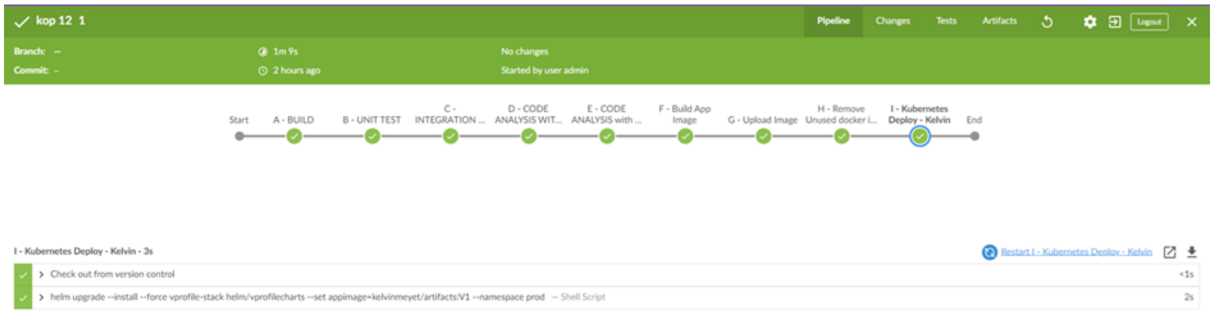
✓ <u>Continuous Integration Continuous Deployment Layer</u>

This layer utilizes GitHub for version control and Jenkins for automating the build, test, and deployment processes. By integrating and configuring plugins such as SonarQube, Maven, Docker to ensure code quality, build automation, containerization, and artifact registry by Docker Hub

GitHub: Acts as the central source code repository, hosting the application code, infrastructure scripts, and Jenkins pipeline configurations (Jenkinsfile). It facilitates version control and collaboration among development teams.

Jenkins: Operates as the CI/CD agent, automating the build, test, and deployment processes. Jenkins pipelines, defined in the Jenkinsfile, integrate with plugins below to enhance functionality.

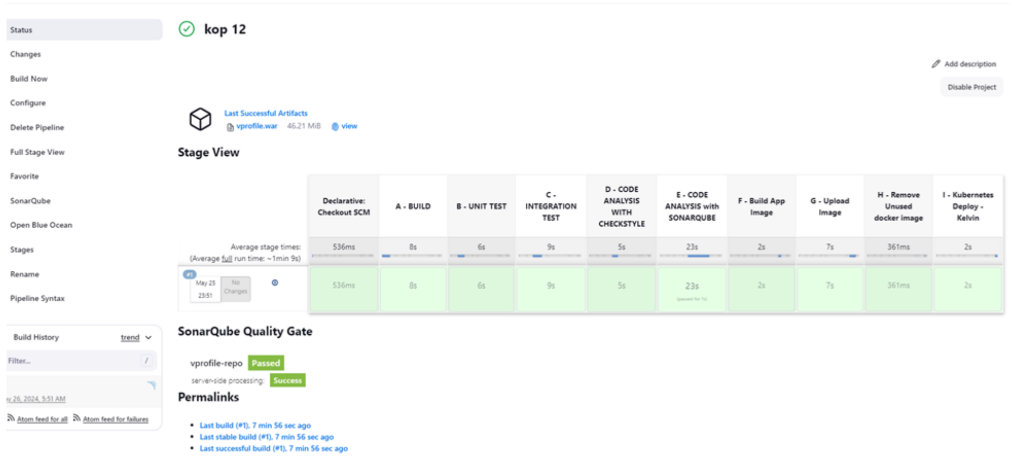
- SonarQube: Performs static code analysis to ensure code quality and adherence to coding standards.
- Pipeline Maven Integration: Manages and builds Java projects, streamlining the build process.
- Pipeline Utility Steps, Blue Ocean, Build Timestamp: Provides enhanced pipeline management and visualization capabilities, improving usability and traceability.
- Docker, Docker Pipeline, Docker API: Enables containerization of applications and efficient management of Docker images throughout the CI/CD pipeline.



Branch: –
Commit: –
1m 9s
2 hours ago
No changes
Started by user admin

I - Kubernetes Deploy - Kelvin - 3s

```
> Check out from version control                                                                          <1s
> helm upgrade --install --force vprofile-stack helm/vprofilecharts --set appimage=kelvinmeyet/artifacts:V1 --namespace prod  — Shell Script   2s
```



kop 12

Last Successful Artifacts
vprofile.war 46.21 MiB view

Stage View

	Declarative: Checkout SCM	A - BUILD	B - UNIT TEST	C - INTEGRATION TEST	D - CODE ANALYSIS WITH CHECKSTYLE	E - CODE ANALYSIS with SONARQUBE	F - Build App Image	G - Upload Image	H - Remove Unused docker image	I - Kubernetes Deploy - Kelvin
Average stage times: (Average full run time: ~1min 9s)	536ms	8s	6s	9s	5s	23s	2s	7s	361ms	2s
#1 May 25 23:51 No Changes	536ms	8s	6s	9s	5s	23s	2s	7s	361ms	2s

SonarQube Quality Gate

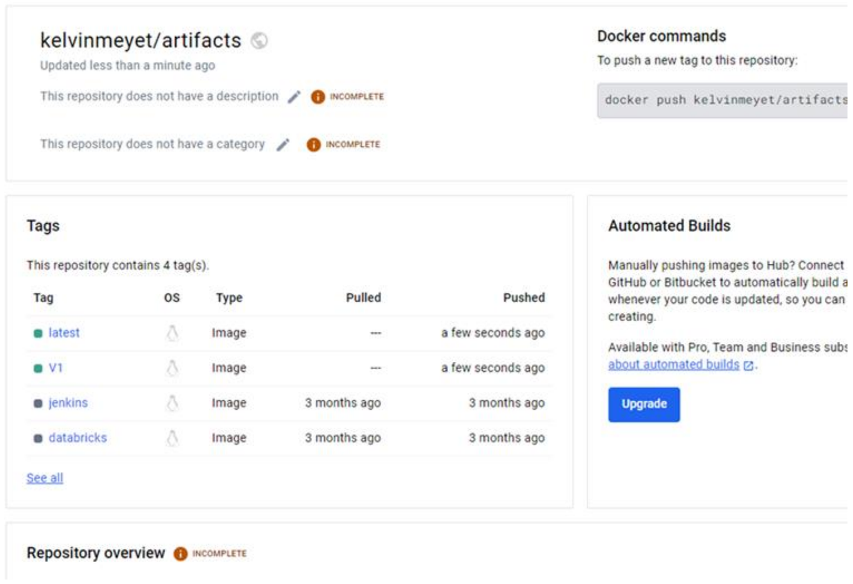
vprofile-repo Passed
server-side processing: Success

Permalinks

- Last build (#1), 7 min 56 sec ago
- Last stable build (#1), 7 min 56 sec ago
- Last successful build (#1), 7 min 56 sec ago

Docker Hub: Serves as the artifact repository, storing Docker images created by Jenkins. These images are subsequently pulled by Kubernetes for deployment, ensuring consistent and reliable containerized applications across different environments.

The integration of these tools facilitates seamless collaboration, efficient build processes, and reliable deployment mechanisms, essential for maintaining high-quality software in an agile development environment.



kelvinmeyet/artifacts
Updated less than a minute ago
This repository does not have a description INCOMPLETE
This repository does not have a category INCOMPLETE

Docker commands
To push a new tag to this repository:

```
docker push kelvinmeyet/artifacts
```

Tags

This repository contains 4 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	---	a few seconds ago
V1		Image	---	a few seconds ago
jenkins		Image	3 months ago	3 months ago
databricks		Image	3 months ago	3 months ago

See all

Automated Builds

Manually pushing images to Hub? Connect GitHub or Bitbucket to automatically build a whenever your code is updated, so you can creating.

Available with Pro, Team and Business subs about automated builds.

Repository overview INCOMPLETE

✓ <u>Application Layer</u>

The application layer is made up of components responsible for executing the core business logic of the e-commerce application and handling dynamic and static content.

Nginx (web server) handles incoming web traffic, serves static content, and forwards requests to the Tomcat application servers. **Tomcat,** being the application server, runs the Java-based e-commerce application by processing business logic and handles dynamic content.

RabbitMQ serves as a message broker, facilitating asynchronous communication between different parts of the application. It's used for tasks like order processing, notifications, and other background jobs.

Kubernetes (Kops & Helm) manages the containerized application deployments. Kops is used to set up and maintain the Kubernetes cluster on AWS, while Helm simplifies and streamlines the deployment and management of applications through Helm charts.

✓ <u>Data Layer</u>

This layer contains components that manage, store and retrieve persistent data required by the e-commerce application.

MySQL: Is the primary relational database management system that stores all persistent data for the application, such as user information, product details, orders, and transactions.

Memcache: Provides a caching layer to speed up database operations by storing frequently accessed data in memory, thereby reducing load on the MySQL database.

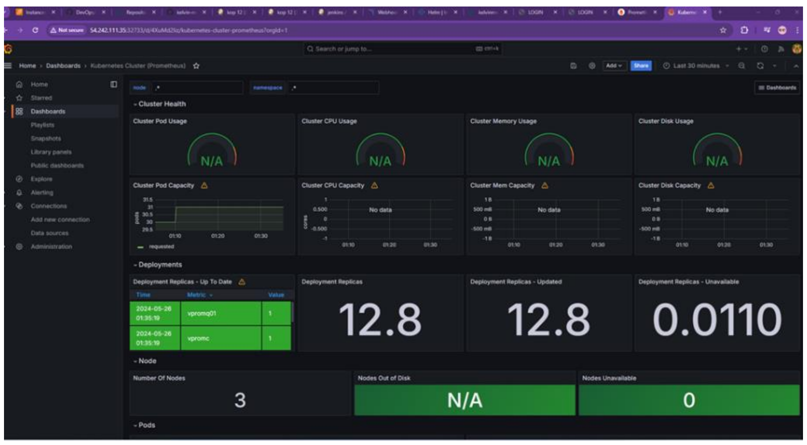
✓ <u>Monitoring and Logging</u>

This layer is responsible for tracking and analyzing performance, health, and behavior of the e-commerce application and its underlying infrastructure. It typically consists of tools and systems designed to collect, store, and visualize various metrics and logs.

Prometheus collects metrics from both application and infrastructure layers.

Grafana visualizes these metrics on dashboards, enabling real-time monitoring and troubleshooting.

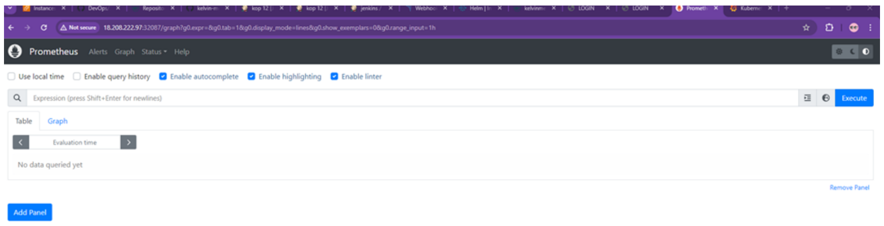
Grafana



Deployment Replicas - Up To Date

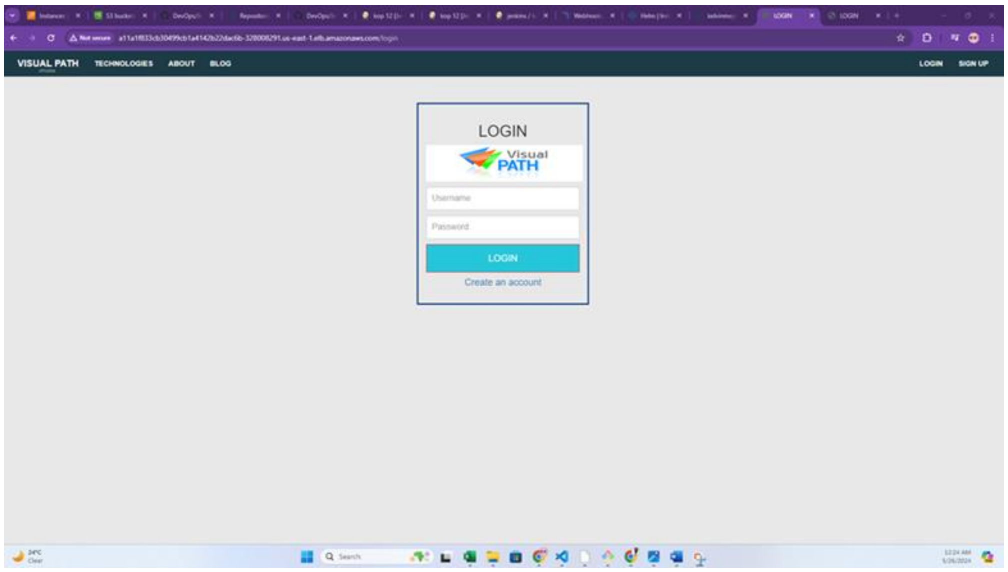
Time	Metric	Value
2024-05-26 01:35:19	vpromq01	1
2024-05-26 01:35:19	vpromc	1

Prometheus



18.208.222.97:32087/graph?g0.expr=&g0.tab=1&g0.display_mode=lines&g0.show_exemplars=0&g0.range_input=1h

E-COMMERCE APPLICATION



a11a1f833cb30499cb1a4142b22dac6b-328008291.us-east-1.elb.amazonaws.com/login

IMPLEMENTING END-TO-END CICD PIPELINE

The system architecture steps have been outlined below in a do it yourself (DIY) style, consisting of two main sections with three subsections in each section.

SECTION 1
Setup Infrastructure, Configure Jenkins with plug-ins, Install Helm charts on Kops Cluster to manage Kubernetes deployment.

1a: Setup Infrastructure

- Start-up Jenkins server, Sonar server and Kops-anchor server on ec2 engines
- Install docker engine on Jenkins server using **install-docker.sh Appendix A**
  Add Jenkins user to the Docker group: $ sudo usermod -aG docker Jenkins
- Sign in to Jenkins server and launch it – ipv4:8080
- Sign in to Sonarqube server and listen on port 80.
  Deploy K8s cluster with Kops (Set – up Kops Cluster on anchor server)
  - Create new user and access keys.
  - Provision K8s cluster with steps named in **Appendix B.**
    - vi install_kops_tool.sh
    - copy & paste bash script in appendix A
    - Run it ($bash install_kops_tool.sh)

1b: Configure Jenkins with Needed Plug-ins and Servers

- On the Jenkins dashboard, Go to Manage Jenkins -> Plug-ins -> Available Plug-ins & search for the following plug-ins to install.
  - ***Sonarqube scanner*** ***Build time stamp*** ***Pipeline Maven Integration***
  - ***Blue Ocean*** ***Pipeline Utility Steps***
  - ***Docker*** ***Docker Pipeline*** ***Docker API***
- On the Jenkins Dashboard -> Manage Jenkins -> Tools
  - JDK installations
    Name = OracleJDK17
    JAVA_HOME = /usr/lib/jvm/java-1.17.0-openjdk-amd64/
  - Maven Installation
    Name = maven3
    Version = 3.9.6
  - SonarQube scanner installation
    Name = mysonarscanner4
    Version = SonarQube scanner 4.7.0.2747 - version 4
  - Save it.

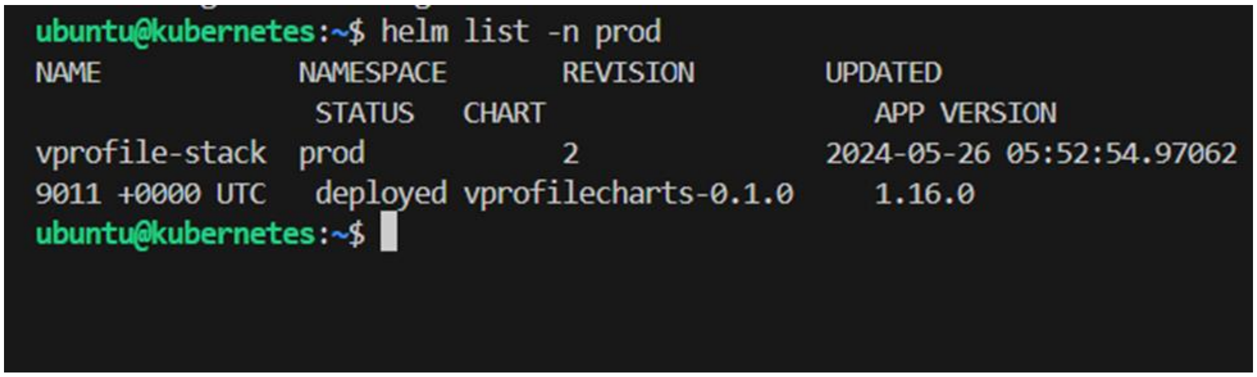
- On the Jenkins Dashboard -> Manage Jenkins -> System
    - Sonarqube servers - > Select environment variables -> Add SonarQube Installation
      Name = sonar
      URL = paste "IPV4 address of sonar server"
      Authenticate server ??? skip
      save it & redo the steps again, it will allow you to authenticate with Jenkins.

    - In the credentials sections: kind = secret text, but secret is in sonarserver so we must log in to sonar server
      sonar login: admin,   pword=admin
      Click on A -> my account -> security
      Give a token name "sonarsecret" & generate it
      copy the token & go back to jenkins and paste the token as secret.
      ID & description = sonartoken
      Save it & choose the authentication.

<u>1c: Install Helm Charts on Kops Cluster to manage K8s Deployment.</u>

(This step allows us to deploy v-profile app on K8s cluster)

- Go to Helm website => www.helm.sh/docs/intro/install
- Go to download desired version & copy link for "linux amd 64"
- Go to kops server and enter the following commands:
    - $wget "paste link of desired version" : Download the helm version
    - $tar -zxvf helm-v3.15.1-linux-amd64.tar.gz : Untars downloaded helm version
    - $sudo mv linux-amd64/helm /usr/local/bin/helm : Moves helm to desired destination
    - $ls /usr/local/bin
    - $helm version
- Clone the Github Repository that contains pipeline codes and specify the branch if needed
    - $git clone "git repo" --branch "branch name" "name on kops server"
- Install Java
    - $sudo apt install openjdk-17-jdk -y
- Create Jenkins slave in opt directory
    - $sudo mkdir /opt/jenkins-slave/
- Change ownership from root to ubuntu
    - $sudo chown ubuntu:ubuntu /opt/jenkins-slave/ -R
    - $ls -ld /opt/jenkins-slave/
- Create namespace to deploy v-profile stack
    - $kubectl create ns prod
    - $kubectl get ns
    - $helm list -n prod



```
ubuntu@kubernetes:~$ helm list -n prod
NAME            NAMESPACE       REVISION        UPDATED
                 STATUS   CHART                    APP VERSION
vprofile-stack  prod            2               2024-05-26 05:52:54.97062
9011 +0000 UTC   deployed vprofilecharts-0.1.0     1.16.0
ubuntu@kubernetes:~$
```

SECTION 2
Configure Jenkins Node with Kops Node, Confirm Pushed Image on Docker Hub, Install Prometheus and Grafana

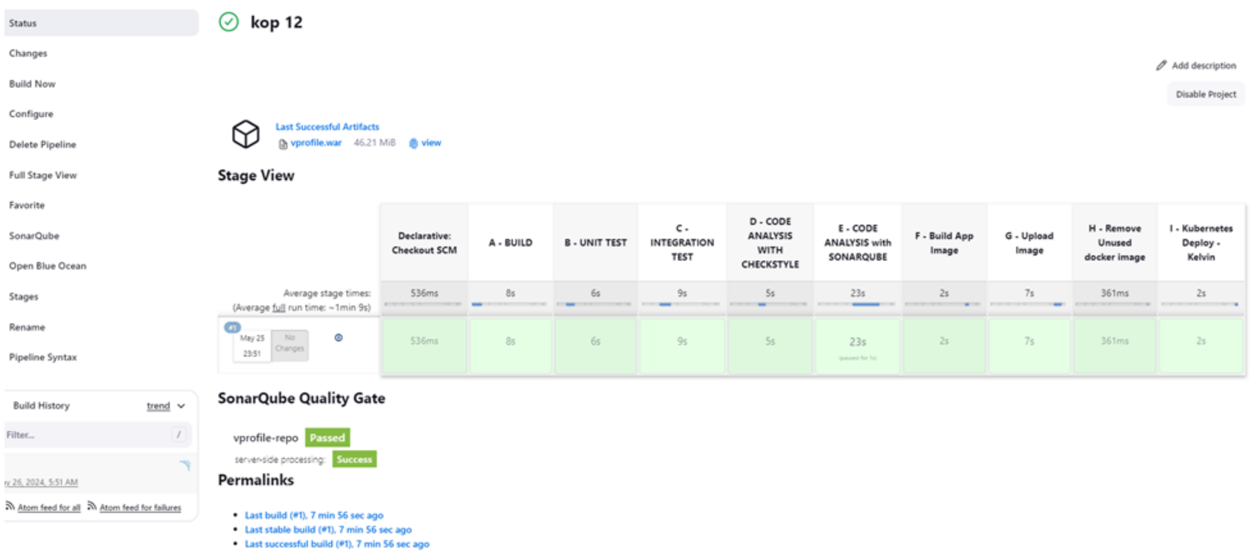
2a: Configure Node on Jenkins Agent with Kops Node to Run CD stage.

- Go to Dashboard -> Manage Jenkins -> Nodes -> New Node
  - Name: Kops
  - Type: permanent agent
  - Remote root dir: /opt/jenkins-slave/
  - Label: KOPS
  - Usage: only build jobs with label expression matching this node
  - Launch method: launch agents via ssh
  - Host = paste IPV4 of kops server
    Credentials: Jenkins
    - Kind: ssh username with private key
    - ID : kops-login
    - Description: kops-login
    - Username: ubuntu
    - Private key: choose enter directly – Copy & paste the private key content (kops.key.pem) & save
    - Choose the ubuntu(kops-login) as credentials
    - Host key verification : Non verifying
    - Availability: Keep this agent online as much as possible
    - Save it
  - Click on Kops -> Logs (check if successful)
- Add docker credentials on Jenkins
  - Dashboard -> Manage Jenkins -> Credentials -> Stored Scoped to Jenkins (System) -> Global credentials -> Add credentials
    - Kind : username with password
    - Username : insert your docker hub username
    - Password: docker hub password
    - ID: dockerhub (as seen on Jenkinsfile)
    - Description: ...
- Webhook on sonarqube (This continues pipeline after passing all tests)
  - Home -> admin -> configuration -> webhook (dropdown) -> create
    - Name : Jenkins-webhook
    - URL: insert jenkins url (http: IPV4:8080/sonarqube-webhook)
    - Create

- Go to Jenkins Dashboard and create Pipeline Job

- Checkout from SCM
- Insert your GitHub Repo URL
- Specify branch.
- Script path = /Jenkinsfile & Build the pipeline
- Check for complete execution and trouble shoot with logs if any issues arise.

COMPLETE CICD PIPELINE



kop 12

Last Successful Artifacts
vprofile.war 46.21 MiB view

Stage View

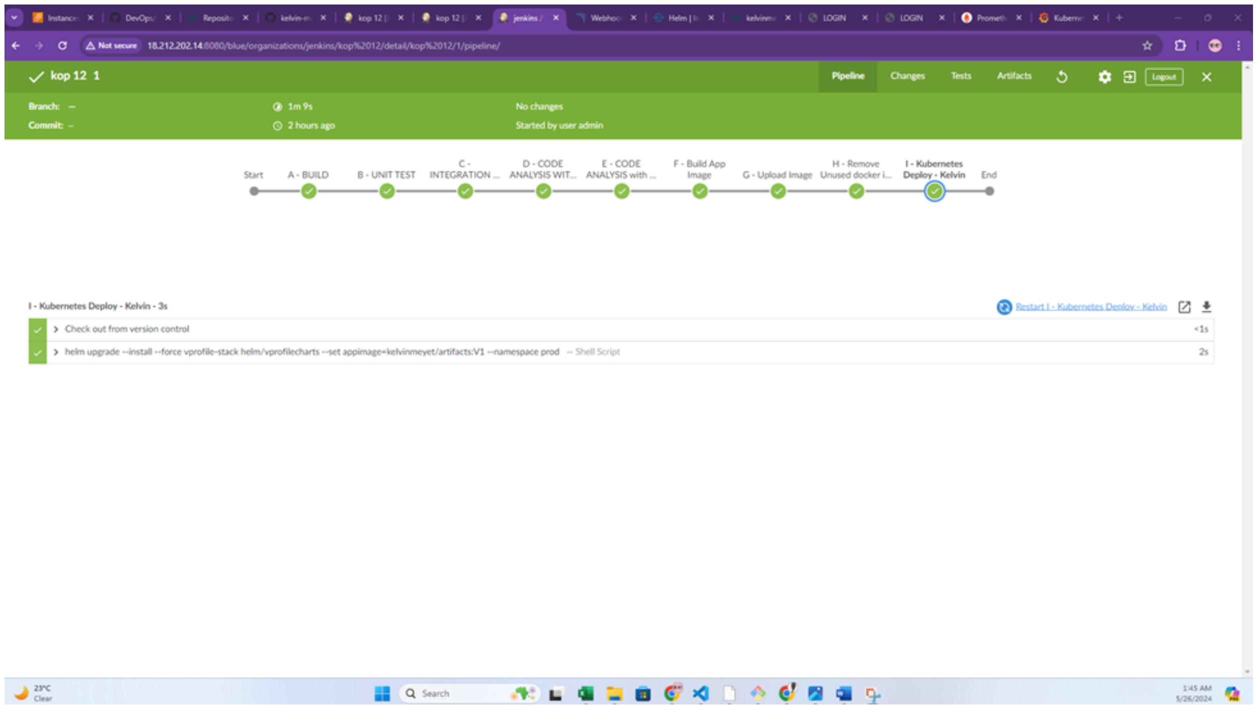
	Declarative: Checkout SCM	A - BUILD	B - UNIT TEST	C - INTEGRATION TEST	D - CODE ANALYSIS WITH CHECKSTYLE	E - CODE ANALYSIS with SONARQUBE	F - Build App Image	G - Upload Image	H - Remove Unused docker image	I - Kubernetes Deploy - Kelvin
Average stage times: (Average full run time: ~1min 9s)	536ms	8s	6s	9s	5s	23s	2s	7s	361ms	2s
#1 May 25 23:51 No Changes	536ms	8s	6s	9s	5s	23s	2s	7s	361ms	2s

SonarQube Quality Gate

vprofile-repo Passed
server-side processing: Success

Permalinks

- Last build (#1), 7 min 56 sec ago
- Last stable build (#1), 7 min 56 sec ago
- Last successful build (#1), 7 min 56 sec ago



18.212.202.14:8080/blue/organizations/jenkins/kop%2012/detail/kop%2012/1/pipeline/

kop 12 1

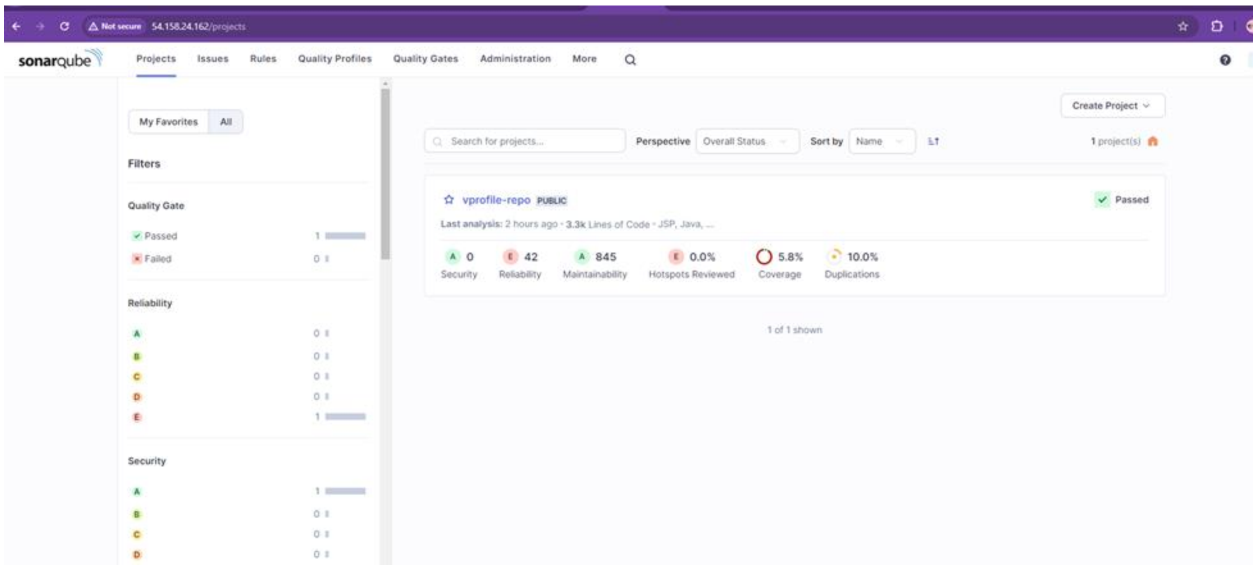
Branch: –
Commit: –
1m 9s
2 hours ago
No changes
Started by user admin

Start — A - BUILD — B - UNIT TEST — C - INTEGRATION ... — D - CODE ANALYSIS WIT... — E - CODE ANALYSIS with ... — F - Build App Image — G - Upload Image — H - Remove Unused docker i... — I - Kubernetes Deploy - Kelvin — End

I - Kubernetes Deploy - Kelvin - 3s

```
> Check out from version control                                                                              <1s
> helm upgrade --install --force vprofile-stack helm/vprofilecharts --set appimage=kelvinmeyet/artifacts:V1 --namespace prod  — Shell Script   2s
```

SONARQUBE CODE SCANNER & QUALITY GATE ASSESSMENT RESULTS



Not secure 54.158.24.162/projects

sonarqube | Projects | Issues | Rules | Quality Profiles | Quality Gates | Administration | More

My Favorites | All

Filters

Quality Gate
- Passed 1
- Failed 0

Reliability
- A 0
- B 0
- C 0
- D 0
- E 1

Security
- A 1
- B 0
- C 0
- D 0

Create Project

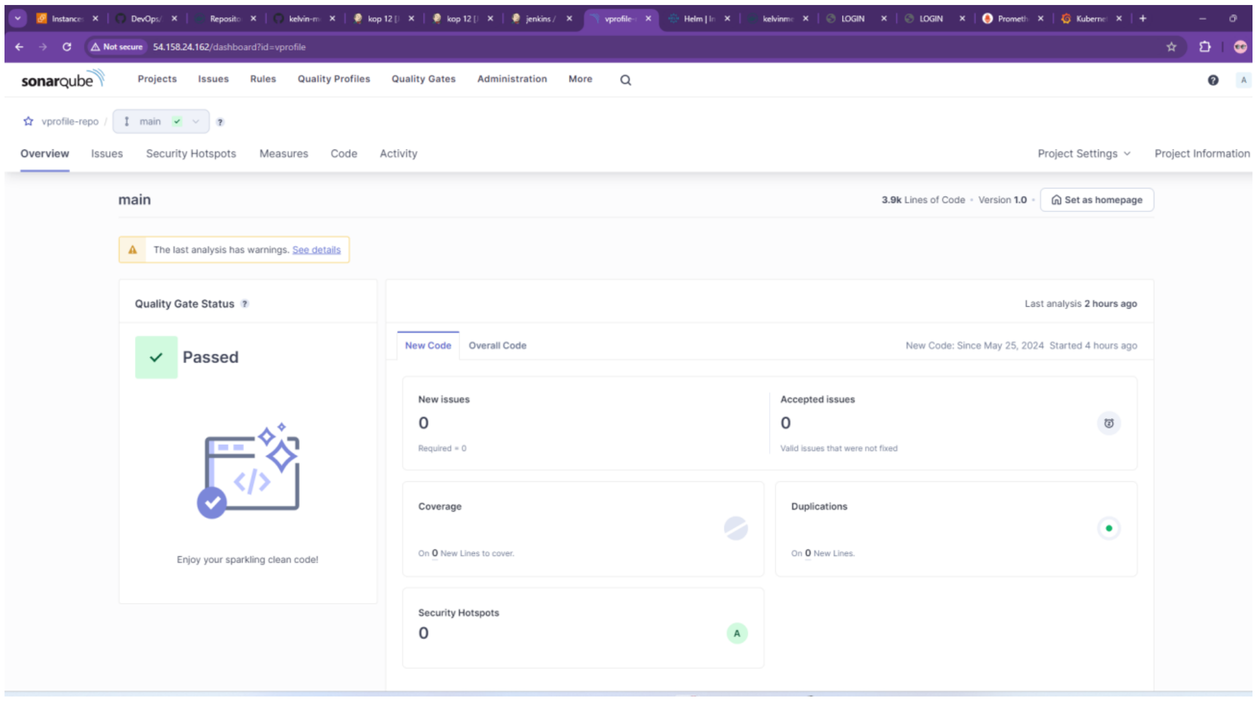
Search for projects... Perspective: Overall Status | Sort by: Name | 1 project(s)

vprofile-repo PUBLIC — Passed

Last analysis: 2 hours ago · 3.3k Lines of Code · JSP, Java, ...

A 0	E 42	A 845	E 0.0%	5.8%	10.0%
Security	Reliability	Maintainability	Hotspots Reviewed	Coverage	Duplications

1 of 1 shown



Not secure 54.158.24.162/dashboard?id=vprofile

sonarqube | Projects | Issues | Rules | Quality Profiles | Quality Gates | Administration | More

vprofile-repo / main

Overview | Issues | Security Hotspots | Measures | Code | Activity | Project Settings | Project Information

main — 3.9k Lines of Code · Version 1.0 · Set as homepage

The last analysis has warnings. See details

Quality Gate Status: Passed

Enjoy your sparkling clean code!

Last analysis 2 hours ago

New Code | Overall Code — New Code: Since May 25, 2024 Started 4 hours ago

New issues: 0 (Required = 0)

Accepted issues: 0 (Valid issues that were not fixed)

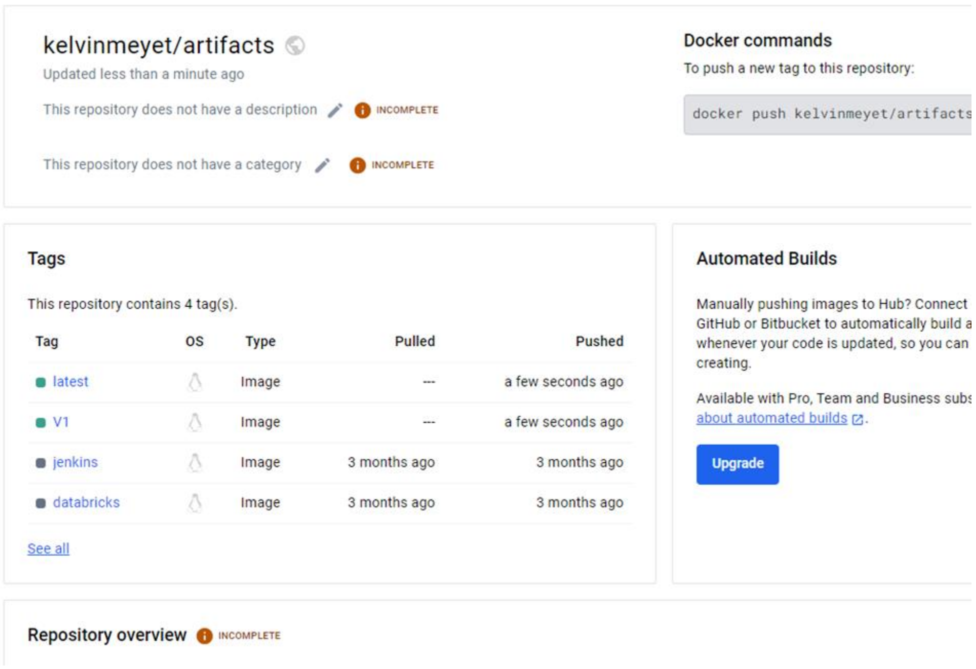
Coverage: On 0 New Lines to cover.

Duplications: On 0 New Lines.

Security Hotspots: 0 — A

2b: Confirm Pushed Image on Docker Hub

- Confirm Image pushed on Docker hub



kelvinmeyet/artifacts

Updated less than a minute ago

This repository does not have a description INCOMPLETE

This repository does not have a category INCOMPLETE

Docker commands

To push a new tag to this repository:

```
docker push kelvinmeyet/artifacts
```

Tags

This repository contains 4 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	---	a few seconds ago
V1		Image	---	a few seconds ago
jenkins		Image	3 months ago	3 months ago
databricks		Image	3 months ago	3 months ago

See all

Automated Builds

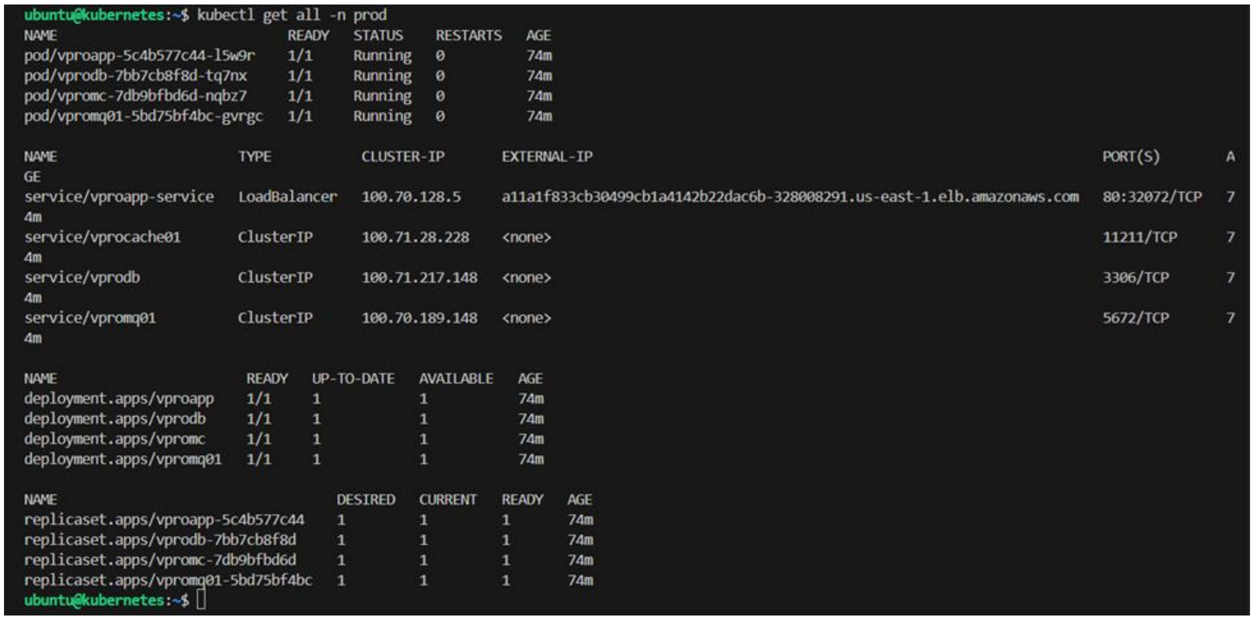
Manually pushing images to Hub? Connect GitHub or Bitbucket to automatically build a whenever your code is updated, so you can creating.

Available with Pro, Team and Business subs about automated builds.

Upgrade

Repository overview INCOMPLETE

- Confirm deployment of v-profile charts
- [$ kubectl get all -n pods] - sees all pods, deployments & replicasets.
    - Copy and paste External IP of LB on browser.



```
ubuntu@kubernetes:~$ kubectl get all -n prod
NAME                              READY   STATUS    RESTARTS   AGE
pod/vproapp-5c4b577c44-l5w9r      1/1     Running   0          74m
pod/vprodb-7bb7cb8f8d-tq7nx       1/1     Running   0          74m
pod/vpromc-7db9bfbd6d-nqbz7       1/1     Running   0          74m
pod/vpromq01-5bd75bf4bc-gvrgc     1/1     Running   0          74m

NAME                         TYPE           CLUSTER-IP       EXTERNAL-IP                                                               PORT(S)        A
GE
service/vproapp-service      LoadBalancer   100.70.128.5     a11a1f833cb30499cb1a4142b22dac6b-328008291.us-east-1.elb.amazonaws.com   80:32072/TCP   7
4m
service/vprocache01          ClusterIP      100.71.28.228    <none>                                                                    11211/TCP      7
4m
service/vprodb               ClusterIP      100.71.217.148   <none>                                                                    3306/TCP       7
4m
service/vpromq01             ClusterIP      100.70.189.148   <none>                                                                    5672/TCP       7
4m

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/vproapp      1/1     1            1           74m
deployment.apps/vprodb       1/1     1            1           74m
deployment.apps/vpromc       1/1     1            1           74m
deployment.apps/vpromq01     1/1     1            1           74m

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/vproapp-5c4b577c44      1         1         1       74m
replicaset.apps/vprodb-7bb7cb8f8d       1         1         1       74m
replicaset.apps/vpromc-7db9bfbd6d       1         1         1       74m
replicaset.apps/vpromq01-5bd75bf4bc     1         1         1       74m
ubuntu@kubernetes:~$
```

- Change Image – [Edit image in deployment.app/vproapp]
  - $kubectl edit deployment vproapp -n prod
  - Vi into file, scroll down to containers section & edit image: use “kubeimran/vproappdock:V3”
  - Make sure image block is parent block of containers
  - Reload the LB page (External IP)

2c: Install Prometheus and Grafana in vproapp namespace

Steps to Install Prometheus:

- $helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
- $ helm search repo prometheus
- $helm repo update

  Create monitoring namespace
- $kubectl create -ns monitoring-tool
- $kubectl get ns
- $ helm install prometheus prometheus-community/kube-prometheus-stack -n monitoring-tools
- $kubectl –namespace monitoring-tool [see all resources for the listening port]
- $kubectl expose service -n monitoring-tool Prometheus-kube-prometheus-prometheus --type=NodePort --target-port=9090 --name=prometheus-server-ext
- $ kubectl get svc -n monitoring-tool [get listening port for Prometheus [30000 – 32000]

  Get instance id for Prometheus
- $kubectl get pods -n monitoring-tool -o wide|grep prometheus-kube-prometheus-prometheus-0|awk '{print $7}'
- Add inbound SG rule for the instance on Prometheus (custom listening port)
- Paste the IPV4:custom port on browser

Steps to Install Grafana:

- $helm repo add grafana https://grafana.github.io/helm-charts
- $ helm search repo prometheus
- $helm repo update
- $helm install grafana grafana/grafana -n monitoring-tool
- $kubectl expose service -n monitoring-tool grafana --type=NodePort --target-port=3000 --name=grafana-ext
- $ kubectl get svc -n monitoring-tool [get listening port for grafana [30000 – 32000]

  Get instance id for Grafana

    - $kubectl get pods -n monitoring-tool -o wide|grep grafana | grep -v prometheus |awk '{print $7}'

      Get Secret for Grafana
    - kubectl get secret –monitoring-tool grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo
    - Add inbound SG rule for the instance on Grafana (custom listening port)
    - Paste the IPV4:custom port on browser
    - Login

      In Grafana, Integrate Prometheus as Data source
    - Datasources -> Prometheus -> Add Prometheus URL -> Save & test
    - Home -> Dashboard -> Import Dashboard -> Enter dashboard ID (3662 or 6417) or URL from grafana website. -> Load - > Select Prometheus from Drop down -> select Default -> Import
- Clean up by following steps in **Appendix D**

Appendix A: Install docker engine on Jenkins server

```
#!/bin/bash


## Update the apt Package Index:

sudo apt-get update

## Install Packages to Allow apt to Use a Repository Over HTTPS:

sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common gnupg lsb-release

## Add Docker’s Official GPG Key:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

## Set Up the Stable Repository:

echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

## Update the apt Package Index (again):

sudo apt-get update

### Install the Latest Version of Docker CE and Containerd:

sudo apt-get install -y docker-ce docker-ce-cli containerd.io

#Add your user to docker group

#sudo usermod -aG docker $USER && newgrp docker

### Verify that Docker CE Is Installed Correctly:

sudo docker run hello-world
```

Appendix B: Install Kops tool on Kops Anchor server

```
#!/bin/bash

# Function to get user input with a verification loop

get_input() {

  local prompt=$1

  local varname=$2

  local input

  while true; do

    echo

    echo "$prompt"

    read input

    echo "You entered: $input"

    echo "Is this correct? (yes/no)"

    read confirmation

    if [[ $confirmation == "yes" ]]; then

      eval $varname="'$input'"

      break

    else

      echo "Please re-enter the information."

    fi

  done

}

# Start of the script logic

get_input "Enter AWS Access Key:" awsaccess

get_input "Enter AWS Secret Key:" awssecret

get_input "Enter VPC ID: (use the VPC ID for the kops instance already created from console)" yourvpcid

get_input "Enter Cluster Name: (ex: my-kube.k8s.local)" clname

get_input "Enter S3 bucket name: (ex: my-kube-k8s-local)" s3buck

get_input "Enter an AZ for the cluster:" az

# Output the entered information (optional, for verification)

echo

echo "AWS Access Key: $awsaccess"

echo "AWS Secret Key: $awssecret"

echo "VPC ID: $yourvpcid"
echo "Cluster Name: $clname"

echo "S3 Bucket Name: $s3buck"

echo "AZ for the cluster: $az"

echo

# Install required packages and utilities

sudo apt update

sudo apt install curl wget awscli -y

# Download kubectl, give execute permission, and move to binary path

curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.25.0/bin/linux/amd64/kubectl

sudo chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl

# Download kOps

sudo curl -LO https://github.com/kubernetes/kops/releases/download/v1.25.0/kops-linux-amd64

# Give executable permission to the downloaded kOps file and move it to binary path

sudo chmod +x kops-linux-amd64

sudo mv kops-linux-amd64 /usr/local/bin/kops

# Ensure the necessary tools are installed

if ! command -v aws >/dev/null 2>&1; then

  echo "AWS CLI is not installed. Please install it first."

  exit 1

fi

if ! command -v kops >/dev/null 2>&1; then

  echo "kops is not installed. Please install it first."

  exit 1

fi

# Configure your AWS user profile

aws configure set aws_access_key_id $awsaccess

aws configure set aws_secret_access_key $awssecret

# Create a key which can be used by kOps for cluster login

if [ ! -f $HOME/.ssh/id_rsa ]; then

  ssh-keygen -N "" -f $HOME/.ssh/id_rsa

else

  echo "SSH key already exists. Skipping creation."

fi

# Create an S3 Bucket where kOps will save all the cluster's state information.
```

```
aws s3 mb s3://$s3buck
```

```
# Expose the s3 bucket as environment variables.

export KOPS_STATE_STORE=s3://$s3buck
```

```
# Create the cluster with 2 worker nodes.

kops create cluster --node-count=2 --master-size="t3.medium" --node-size="t3.medium" --master-volume-size=30 --node-volume-size=30 --zones=$az --name $clname --state=s3://$s3buck --vpc=$yourvpcid
```

```
# Apply the specified cluster specifications to the cluster

kops get cluster

kops update cluster $clname --yes --state=s3://$s3buck
```

```
# The .bashrc file is a script file that’s executed when a user logs in.

echo "export KOPS_STATE_STORE=s3://$s3buck" >> ~/.bashrc
```

APPENDIX D: Clean up

Go to the Kops Anchor server

```
kops delete cluster --name=<cluster-name>.k8s.local --yes
```

delete the bucket manually from the console including contents

Exit from each server and destroy each terraform infrastructure with

```
exit

Terraform destroy - -auto -approve
```

APPENDIX 00: Alternative way to install helm charts on Kops Cluster

Since this Prometheus instance is designed to monitor a Kubernetes cluster, you must first provision the cluster using kOps. You can find the detailed instructions [here](https://github.com/techlearn-center/DevOps/blob/CICD/Kubernetes/kOps.md) .

- Installing and configuring Helm:

Download the binaries, extract it and put it in /usr/local/bin/helm.

Change the directory to /tmp (this directory is designed for temporary file storage. Since it's often used for intermediate files during software installation or processes that don't require long-term storage,it's a convenient place to download and unpack binaries that you might not need to keep permanently):

```
cd /tmp
```

To install the binaries, download them using the following command:

```
wget https://get.helm.sh/helm-v3.15.1-linux-amd64.tar.gz
```

Extract the binary with the following command:

```
tar xzvf  helm-v3.14.4-linux-amd64.tar.gz
```

We now have the Helm binary, and we will move it to `/usr/local/bin/helm`.

```

sudo mv /tmp/linux-amd64/helm /usr/local/bin/helm

```

```
cd
```

```
helm –help
```